NaWrapper

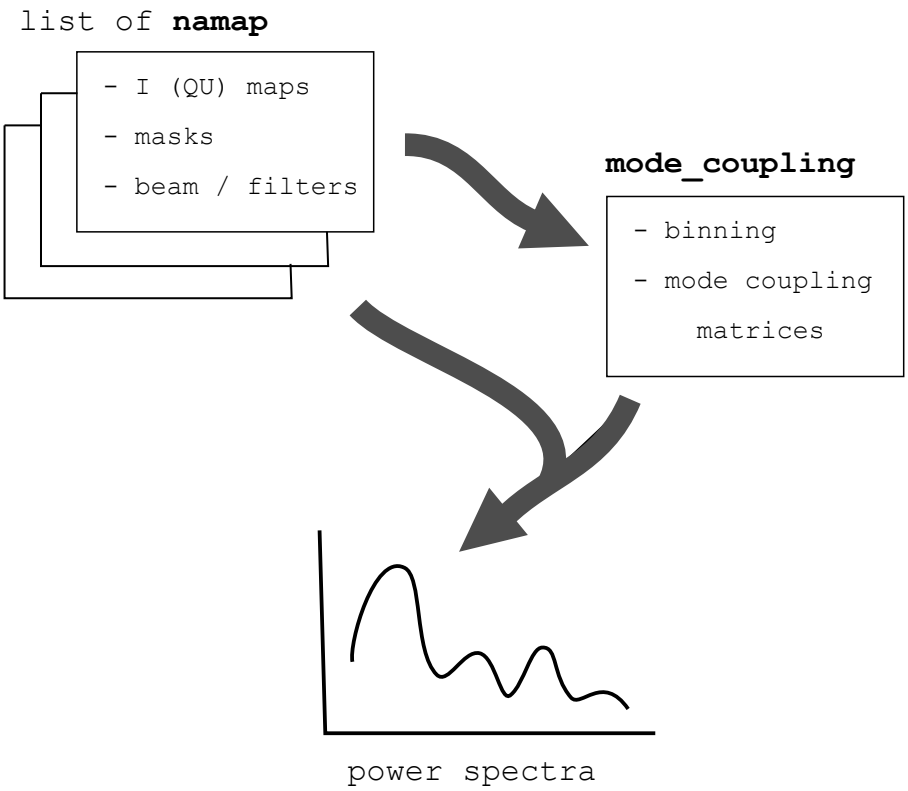
Release 0.1

Zack Li

Aug 06, 2019

CONTENTS

NaWrapper is a thin python wrapper around NaMaster for CMB work in rectangular pixelization, in particular for the ACT experiment.



It bundles all the necessary pieces associated with a map (i.e. beams, masks, filters) into a `nawrapper.ps.namap` object, and provides concise utility functions for turning these into dictionaries of power spectra.

CHAPTER

ONE

INSTALLATION

This guide is intended for people who are not currently using Anaconda. If you already have a Python 3 conda environment, skip below to the subsection *Installing Software*.

1.1 Create a Conda Environment

You first need to get the Anaconda module and enter a custom environment.

```
module load anaconda3
```

Either enter your preferred conda environment with `source activate YOUR_ENV_NAME` if you already have one, or create a new one. I call my power spectrum conda environment `ps`, but you can name it something else if you want, just replace every mention of `ps` in the instructions below with your desired environment name.

```
conda create -n ps python=3
```

Enter your environment. This is what you will type every time you want to compute some spectra (and in your SLURM power spectrum jobs!).

```
conda activate ps
```

You will need to install some other necessary packages into your conda environment, if you just created it. Note that we are using `-n` to specify the conda environment!

```
conda install -n ps matplotlib cython numpy scipy astropy pillow
```

If you are running on a machine with Jupyter like cori or niagara, we also set up ipykernel so your conda environment shows up in Jupyter. You can do this on tiger too, but it's less convenient (see Jupyter on the Cluster).

```
conda install -n ps ipykernel
python -m ipykernel install --user \
    --name ps --display-name "Python 3 (ps)"
```

1.2 Installing Software

We now install the power spectrum software. Install NaMaster with

```
conda install -c conda-forge namaster -n ps
```

You also need to install pixell. This involves cloning a GitHub repository, so for convenience you might want to set up your SSH keys on GitHub. If you are on niagara, you will also need to `module load autotools` to get autoconf before installing pixell. Go to the directory where you keep your software, and run

```
git clone git@github.com:simonsobs/pixell.git
cd pixell
python setup.py install --user
```

Then install nawrapper (these routines for ACT power spectrum analysis). Return to the directory where you keep your software, and then run

```
git clone git@github.com:xzackli/nawrapper.git
cd nawrapper
pip install -e . --user
```

You should be all set! Try out the *Quickstart*.

1.2.1 Updating NaWrapper

To get new version of NaWrapper in the future, you'll have to run *git pull* in the *nawrapper* repository you had cloned earlier. Since you installed with *-e* using pip, this is all you have to do.

CHAPTER

TWO

QUICKSTART

2.1 Workflow Tips

Although this package works perfectly fine on a laptop, you should probably do your science analysis on the cluster. A set of four standard ACT resolution split maps takes ~5 minutes to cook into spectra on a 40-core tiger node. The same on your laptop would take more than three hours!

Some clusters have Jupyter support, but we've found it nice to run this quickstart tutorial with *ipython*. You might need to install it to your conda environment.

```
conda install ipython -n YOUR_ENV_NAME
```

To get your plots to appear on your laptop, make sure to tunnel in with *ssh -Y*.

NaMaster also depends on the environmental variable *OMP_NUM_THREADS* to determine how many threads to run in parallel. On tiger, I run

```
export OMP_NUM_THREADS=40
```

before running any code. You should set this, with the number corresponding to the number of cores on your machine.

2.2 Simulated Maps

In this example, we generate some fake spectra plagued by bright point sources. We then mask those sources and generate spectra which have been corrected for the mode coupling induced by our mask. This example is also available as a Jupyter notebook.

We start by importing our libraries.

```
import nawrapper.ps as nw
import pymaster as nmt
import numpy as np
import matplotlib.pyplot as plt
from pixell import enmap, enplot
```

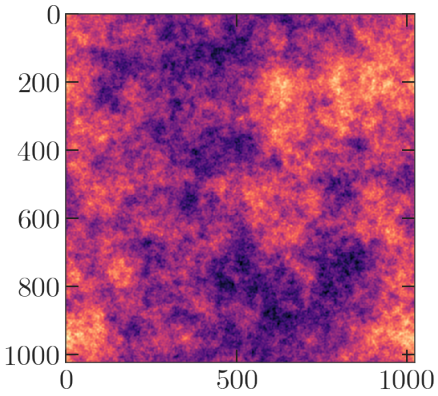
Next, we generate a random map realization for the spectrum $C_\ell = \ell^{-2.5}$.

```
# map information
shape,wcs = enmap.geometry(shape=(1024,1024),
    res=np.deg2rad(0.5/60.),pos=(0,0))

# create power spectrum information
```

(continues on next page)

(continued from previous page)

```
ells = np.arange(0,6000,1)
ps = np.zeros(len(ells))
ps[2:] = 1/ells[2:]**2.5 # don't want monopole/dipole

# generate a realization
imap = enmap.rand_map(shape,wcs,ps[np.newaxis, np.newaxis])
plt.imshow(imap)
```



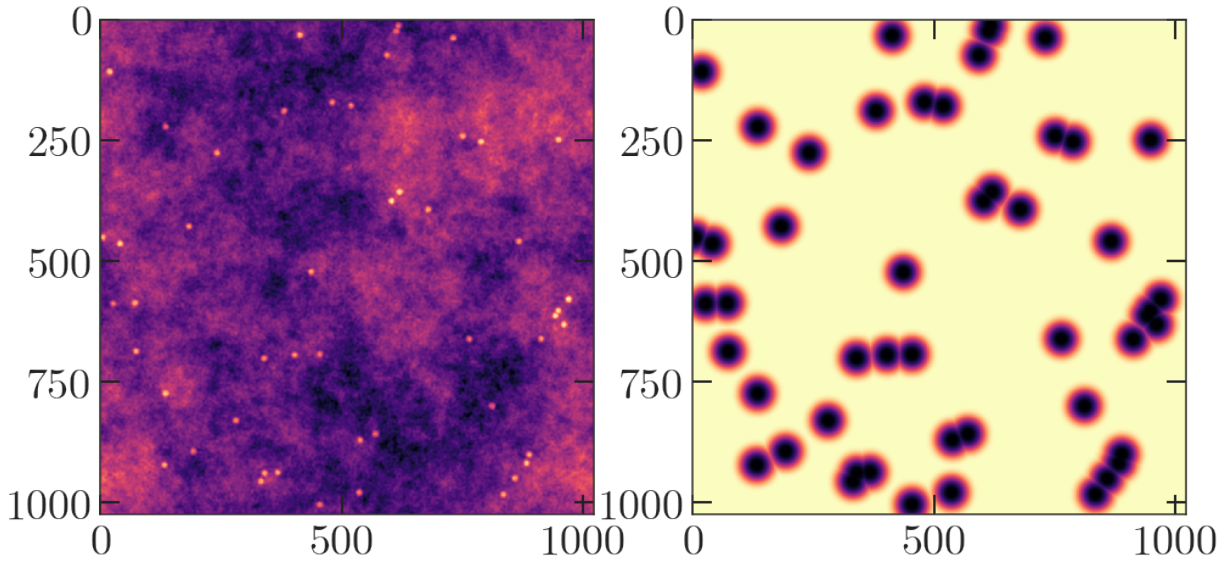
Next, we generate a fake point source map and a matching source mask.

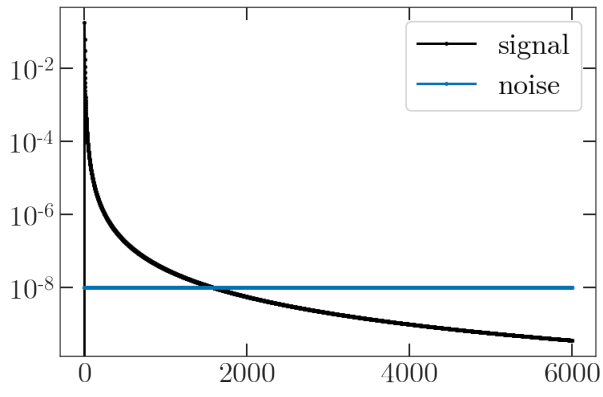
```
mask = enmap.ones(imap.shape, imap.wcs)

N_point_sources = 50
for i in range(N_point_sources):
    mask[
        np.random.randint(low=0, high=mask.shape[0]),
        np.random.randint(low=0, high=mask.shape[1]) ] = 0
# apodize the pixels to make fake sources
point_source_map = 1-nw.apod_C2(mask, 0.1)

imap += point_source_map # add our sources to the map
mask = nw.apod_C2(mask, 0.5) # apodize the mask

# plot our cool results
fig, axes = plt.subplots(1, 2, figsize=(8,16))
axes[0].imshow(imap)
axes[1].imshow(mask)
```

For additional realism we generate noise power spectra to add to our "splits".

```
ells = np.arange(0,len(ps),1)
nl = np.ones(len(ells)) * 1e-8

noise_map_1 = enmap.rand_map(shape,wcs,nl[np.newaxis, np.newaxis])
noise_map_2 = enmap.rand_map(shape,wcs,nl[np.newaxis, np.newaxis])

plt.plot(ps, label="ps")
plt.plot(nl, label="noise")
plt.yscale('log')
plt.legend()
```

For this example, we won't include a beam. Now we set up the `nawrapper.ps.namap` objects, using as input our


0
250
500
750
1000
0
500
1000
0
250
500
750
1000
0
500
1000


signal
noise
10^{-2}
10^{-4}
10^{-6}
10^{-8}
0
2000
4000
6000

our original random realization summed with the noise realizations.

```
namap_1 = nw.namap(map_I=imap + noise_map_1, mask=mask)
namap_2 = nw.namap(map_I=imap + noise_map_2, mask=mask)
```

Next we compute the mode-coupling matrix. We need the binning file, which we have in the repository under *notebooks/data/*. You'll need to point it to the right path on your own installation. This is the slow part, you might want to get a snack while you wait (about 5 minutes on 1 core for a map of this size).

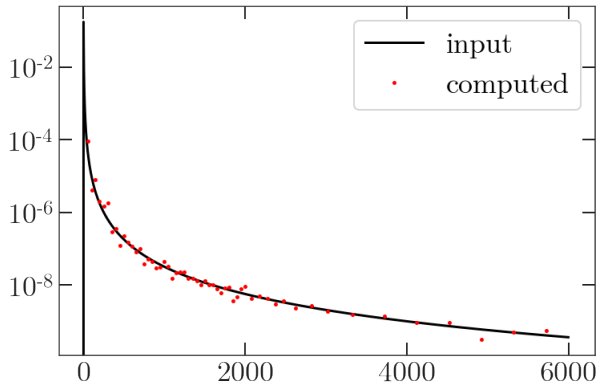
```
binfile = 'data/BIN_ACTPOL_50_4_SC_low_ell'
bins = nw.read_bins(binfile)
mc = nw.mode_coupling(namap_1, namap_2, bins)
```

Finally, we can compute some spectra! Pass in the namaps we created, with the mode coupling object.

```
Cb = nw.compute_spectra(namap_1, namap_2, mc=mc)
```

Let's plot it!

```
plt.plot(ps, 'k-', label='input')
plt.plot(Cb['ell'], Cb['TT'], 'r.', label='computed')
plt.legend()
plt.yscale('log')
```



We've recovered our input spectrum!

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search